EECS 214 Spring 2016 Worksheet 1

Stacks

In a sentence, describe what a stack does at a high level. Namely, how does a stack differ from a queue?

- Answer: look for “last in/first out” or “first in/last out”, LIFO, etc.

For each of these stack operations, explain what they do and give the worst-case O(n) complexity.

- **Push**
- **Pop**
- **Peek**

Answers:

- Push: O(1), insert value at top of stack
- Pop: O(1), remove and return value from top of stack
- Peek: O(1), return top value from stack without removing it

For the rest of the section, assume the following class definition for a stack implemented using an array:

```
public class Stack
    {
        // Define object properties here
        object[] stack = new object[10];
        int first = 0;
        int last = 0;
        int size = 0;

        public override void Push(object o)
        { ... }

        public override object Pop()
        { ... }

        public override int Count
```

```
        { ... }

        public override object Peek()
        { ... }

        public override bool IsFull
        { ... }
    }
```

Assume you have initialized the following stack, `MyStack` :

```
Stack myStack = new Stack();
myStack.Push(3);
myStack.Push(1);
myStack.Push(-5);
myStack.Push(0);
myStack.Push(2);
```

What is the expected return value of each of the following operations? (Note that some operations will not have a return value; in this case, just write `void`)

```
myStack.Pop();    // Answer: 2
myStack.Push(4);  // Answer: Void
myStack.IsFull;   // Answer: False
myStack.Pop();    // Answer: 4
myStack.Pop();    // Answer: 0
myStack.Peek();   // Answer: -5
myStack.Count;    // Answer: 3
myStack.Pop();    // Answer: -5
```

The following exercise will prepare you to write unit tests for a stack implementation.

For each stack operation, your goal is to comprehensively describe the operation's expected behavior in terms of example inputs and expected outputs. Then, use `Assert` to write each example as a valid unit test. Here is an example:

```
[TestMethod]
public void PushTest() {
```

```
    // Set up any structures you need for the test
    Stack myStack = new Stack();
    myStack.Push(4);
    myStack.Push(3);
    // Use assertions to test expected behaviors
    Assert.AreEqual(3, myStack.Pop());
}
```

You may use the initial definition of `MyStack` (before the operations in the previous question), and you may also define new `Stacks` for testing. Be sure to write enough examples to cover edge cases comprehensively (for example, consider how an operation should behave when a `Stack` is full).

- **Push**

Should add the new value to the top of the stack

```
// MyStack: 3, 1, -5, 0, 2
MyStack.Push(6);
// MyStack: 6, 3, 1, -5, 0, 2
```

Should throw an exception if stack is full

```
// FullStack: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
FullStack.Push(6);  // throws StackFullException
```

- **Pop**

Should remove the topmost value and return it

```
// MyStack: 3, 1, -5, 0, 2
MyStack.Pop();  // returns 2
// MyStack: 3, 1, -5, 0
```

Should throw an exception if stack is empty

```
// EmptyStack:
EmptyStack.Pop();  // throws StackEmptyException
```

- **Count**

Should return the correct size for a nonempty stack

```
// MyStack: 3, 1, -5, 0, 2
MyStack.Count;  // returns 5
```

Should return 0 for an empty stack

```
// EmptyStack:
EmptyStack.Count;  // returns 0
```

- **IsFull**

Should return `false` when stack is empty and not full

```
// MyStack: 3, 1, -5, 0, 2
MyStack.IsFull;  // returns false

// EmptyStack:
EmptyStack.IsFull;  // returns false
```

Should return `true` when stack is full

```
// FullStack: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
FullStack.IsFull;  // returns true
```

Queues

In a sentence, describe what a Queue does at a high level

- Answer: Key words: “First in First Out” (FIFO), new entries go to back of line, etc.

For each of the following Queue operations explain what they do and give the worst-case O(n) complexity.

- **Insert/Enqueue**
- **Remove/Dequeue**

Answer:
Insert- place value at end of queue
Remove- take and store value from front of queue

O(1), O(1)

For the following questions assume the following implementation of a Queue

```

public class Queue
    {
        // Define object properties here
```

```
        object[] queue= new object[10];
        int first = 0;
        int last = 0;
        int size = 0;

        public override void Enqueue(object o)
        { ... }

        public override object Dequeue()
        { ... }

        public override int Count
        { ... }

        public override bool IsFull
        { ... }
        //checks to see if the Queue does not have more room
    }
```

Assume that we have created a Queue, MyQueue, via the following operations

```
Queue MyQueue = new Queue ();
//created our Queue
MyQueue.Enqueue(10);
MyQueue.Enqueue(20);
MyQueue.Enqueue(30);
MyQueue.Enqueue(40);
MyQueue.Enqueue(50);
```

What is the expected return value of the following operations? (similar to the Stack question, you may write void for a void return, however note that these operations still affect our queue)

```
MyQueue.Enqueue(100); //Void
MyQueue.Count(); //6
MyQueue.Dequeue(); //10
```

```
MyQueue.Enqueue(25); //Void
MyQueue.Dequeue(); //20
MyQueue.IsFull(); //False
```

Similar to the Stack test cases, write test cases that would ensure that our Queue implementation is working properly. You can use MyQueue from above (before we changed it)

Answer code:

```
[TestMethod]
public void QueuingTests() {


  Assert.AreEqual(10, MyQueue.Dequeue()); // this checks to see if the first dequeue would work
  int count = MyQueue.Count();
  MyQueue.Enqueue(40);
  Assert.AreNotEqual(count, MyQueue.Count());



}


[TestMethod]
public void CountTests() {
  Assert.AreEqual(5, MyQueue.Count()); //testing count
  Assert.IsFalse(MyQueue.IsFull()); //Making sure right now isfull is working
  //making the queue full so we can properly test isfull
  for (int i = 1; i<5; i++){
  MyQueue.Enqueue(i);
  }
  Assert.IsTrue(MyQueue.IsFull());  //should be full now
}


```

```
25 [TestMethod]
26 //this line deals with exceptions, even if you want them like in this c
   ase
27 [ExpectedException(typeof(ArgumentException),
28     "The Queue is properly creating an exception for a full queue")]
29 public void FullTest() {
30   MyQueue.Enqueue(10); //This should throw an exception thus testing be
   ing able to       check if the queue is full
31 }
32
33
```